Árvore Binária de Pesquisa

- Construtor da classe `CArvoreBinaria`

```
template <class T> CArvoreBinaria<T>::CArvoreBinaria(){
   raiz = nullptr;
}
```

- Destrutor da classe `CArvoreBinaria`

```
template <class T> CArvoreBinaria<T>::~CArvoreBinaria(){
   destroi();    // método que elimina todos os nós
}
```


Árvore Binária de Pesquisa

- Métodos para destruir parcial ou totalmente a árvore

```
template <class T> void CArvoreBinaria<T>::destroi(void) {
   destroiSubArvore(raiz);
   raiz = nullptr;
}
//-------------------------------------------------------
template <class T> void CArvoreBinaria<T>::destroiSubArvore(
                        CNoArvoreBinaria<T> *pArvore) {
   if (pArvore != nullptr) {
      destroiSubArvore(pArvore->esq);
      destroiSubArvore(pArvore->dir);
      delete pArvore;
   }
}
```

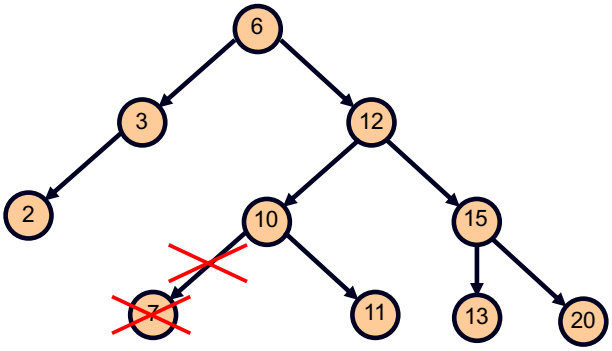
Árvore Binária de Pesquisa

- Eliminação de um nó:
  - Operação mais complexa do que a inserção porque às vezes implica reorganizar toda a árvore para manter a ordem correta dos nós restantes
  - Há 3 casos a considerar:
    1. Eliminação de uma folha
    2. Eliminação de um nó com um filho único
    3. Eliminação de um nó com dois filhos (o caso mais complexo)
  - Em qualquer caso, é necessário pesquisar o nó na árvore antes de proceder à sua eliminação

Árvore Binária de Pesquisa

1. Eliminação de uma folha

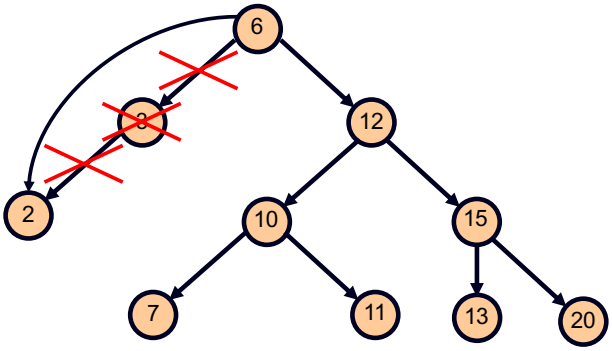
- Fácil de resolver
- Basta remover o nó e atribuir ao ponteiro do seu nó pai o valor `nullptr`
- Exemplo – remoção do 7



Árvore Binária de Pesquisa

2. Eliminação de um nó com um filho único

- Fácil de resolver
- Basta remover o nó e substituí-lo pelo seu filho
- Exemplo – remoção do 3

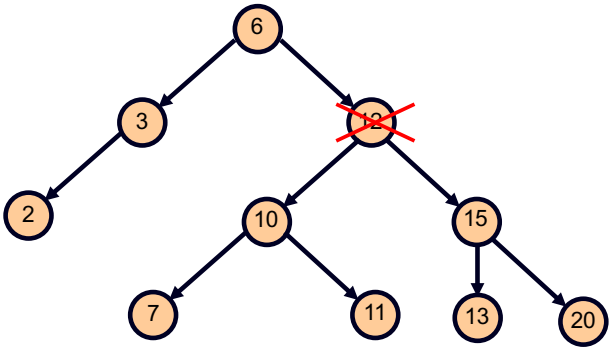


Árvore Binária de Pesquisa

3. Eliminação de um nó com dois filhos

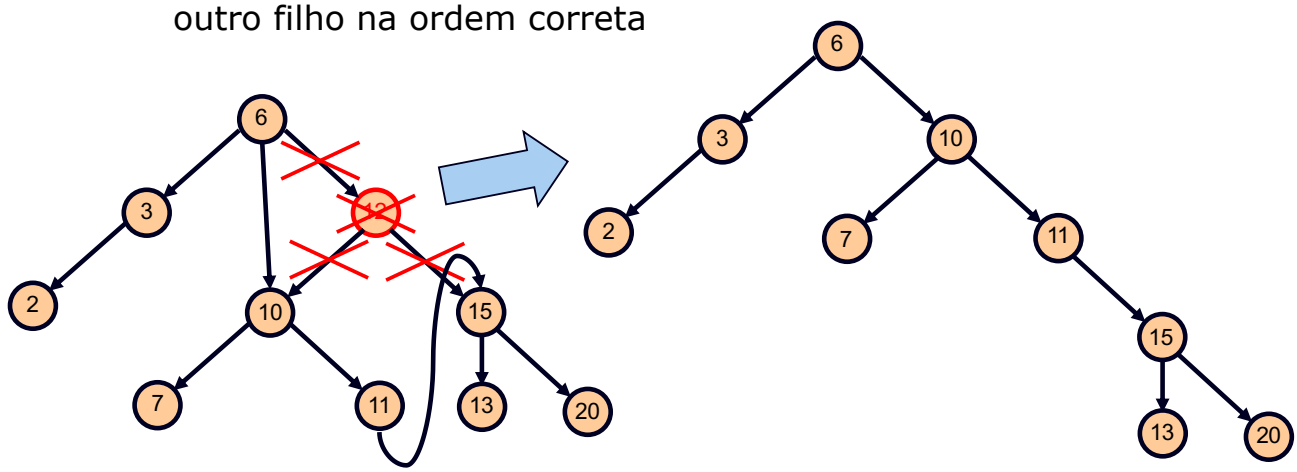
- Ambas as subárvores filhas do nó a eliminar têm de ser religadas de forma correta
- Duas soluções
  a) Substituir o nó a eliminar por um dos seus filhos e depois religar o outro filho na ordem correta
  b) Substitui-lo pelo menor dos seus descendentes maiores do que ele

- Exemplo: remoção do 12



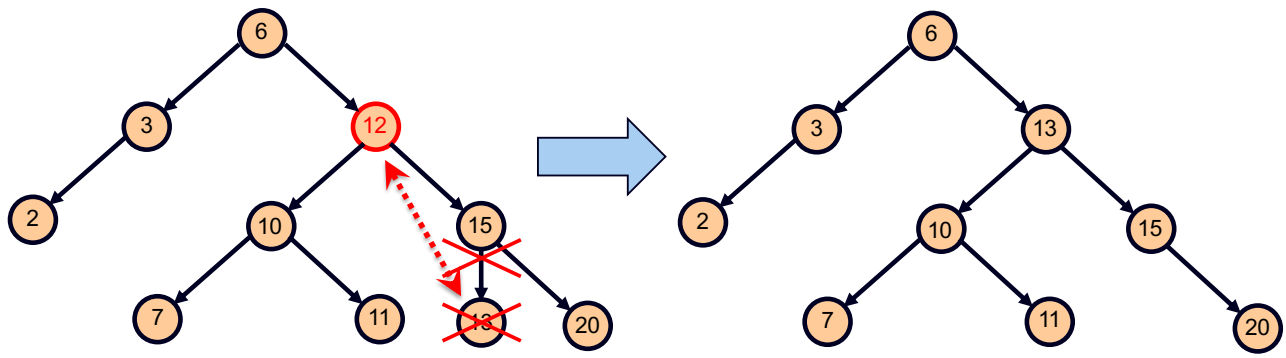
Árvore Binária de Pesquisa

3. Eliminação de um nó com dois filhos
   a) Substituir o nó a eliminar por um dos seus filhos e depois religar o outro filho na ordem correta



Árvore Binária de Pesquisa

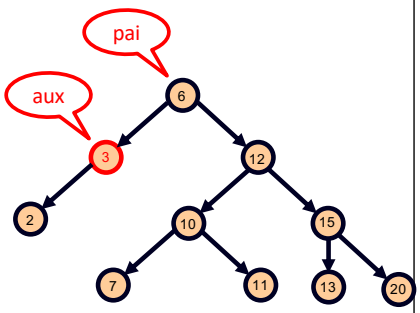
3. Eliminação de um nó com dois filhos
   b) Substituir o nó a eliminar pelo menor dos seus descendentes maiores do que ele



Árvore Binária de Pesquisa

- Eliminação de um nó da árvore (procura nó)

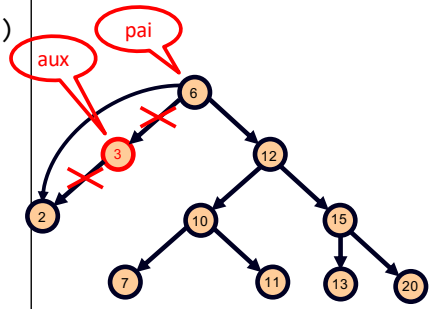
```
template <class T>
bool CArvoreBinaria<T>::elimina(const T& item) {
   CNoArvoreBinaria<T> *pai, *aux, *filho, *sucessor;

   bool encontrado=false;

   // procura o nó a eliminar
   pai = nullptr; aux = raiz;
   while ( (aux != nullptr) && (!encontrado) ) {
      if (aux->dados == item) encontrado = true;
      else {
              pai = aux;
              if (item < aux->dados) aux = aux->esq;
                  else aux = aux->dir;
      }
   }
   // se for encontrado: 'aux' aponta para o nó a eliminar, 'pai' para o seu pai
   ...
```



Árvore Binária de Pesquisa

- Eliminação de um nó da árvore (casos 1 e 2)

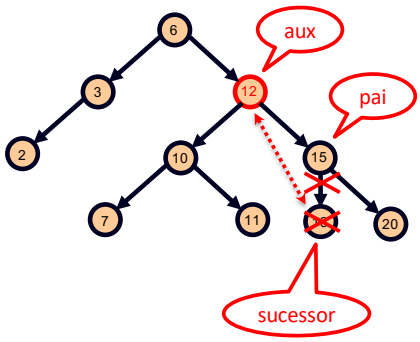
```
   ...
   if (encontrado) {
      // se nó tem no máximo um filho (casos 1 e 2)
      if ((aux->esq == nullptr) || (aux->dir == nullptr))
      {
         if (aux->esq != nullptr) filho = aux->esq;
         else filho = aux->dir; // pode ser nullptr
         if (pai == nullptr)
            raiz = filho; // caso especial: apaga raiz
         else { // atualiza ligações
            if (aux->dados < pai->dados)
               pai->esq = filho;
         else
               pai->dir = filho;
         }
         delete aux;
      }
      ...
```



Árvore Binária de Pesquisa

- Eliminação de um nó da árvore (caso 3)

```
        ...
        else { // elimina nó com dois filhos (caso 3)
           // procura sucessor
           sucessor = aux->dir; pai = aux;
           while (sucessor->esq != nullptr) {
              pai = sucessor; sucessor = sucessor->esq;
           }
           aux->dados = sucessor->dados; // troca valor
           if (sucessor == aux->dir)
              aux->dir = sucessor->dir;
           else // substitui pelo seu sucessor à direita
              pai->esq = sucessor->dir;
           delete sucessor;
        }
        return true; // encontrado
     }
     return false; // não encontrado
  }
```

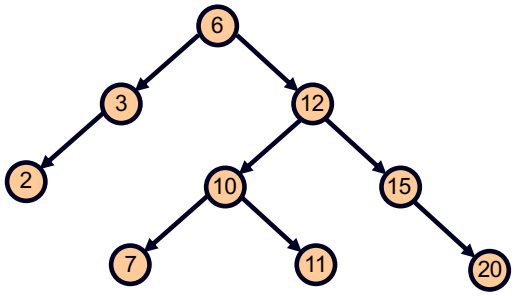


5. Árvores Binárias de Pesquisa

- Qual o interesse das estruturas em árvore?
- Árvores – conceitos gerais
- Árvores binárias
- Programação de uma classe de objetos para manipular árvores binárias de pesquisa
- Tópicos e exercícios adicionais

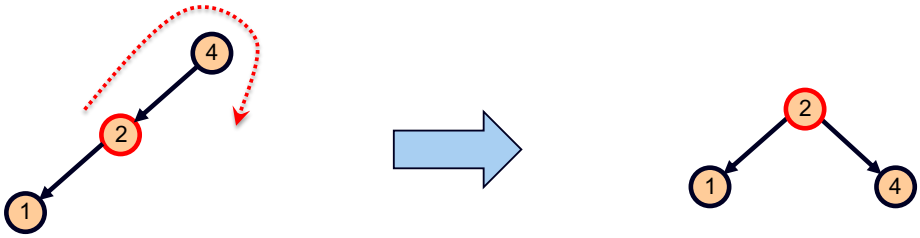
Árvore AVL

- Árvore de Adel'son-Vel'skii & Landis (1962)
  - Árvore equilibrada: em cada nó, a altura das suas subárvores difere no máximo de um
  - Se uma operação de inserção ou remoção de nós puser em causa esta propriedade, tem lugar a reorganização da árvore



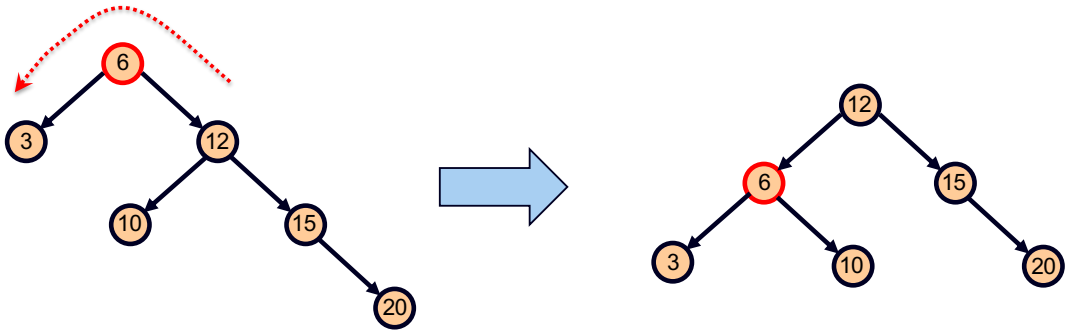
Árvore AVL

- Equilibrar uma árvore por rotação em torno de um nó pivô



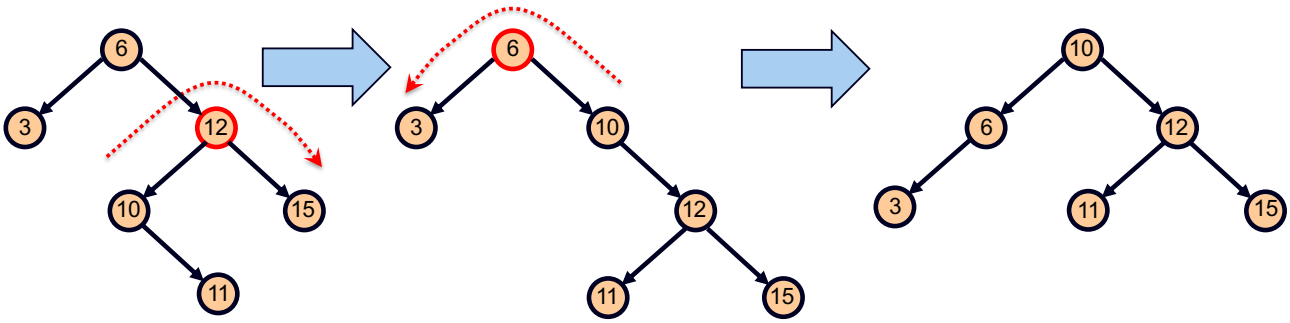
Árvore AVL

- Equilibrar uma árvore por rotação em torno de um nó pivô:



Árvore AVL

- Por vezes, uma rotação não é suficiente para equilibrar a árvore



- Implementação deste tipo de operações: fora do âmbito da unidade curricular...

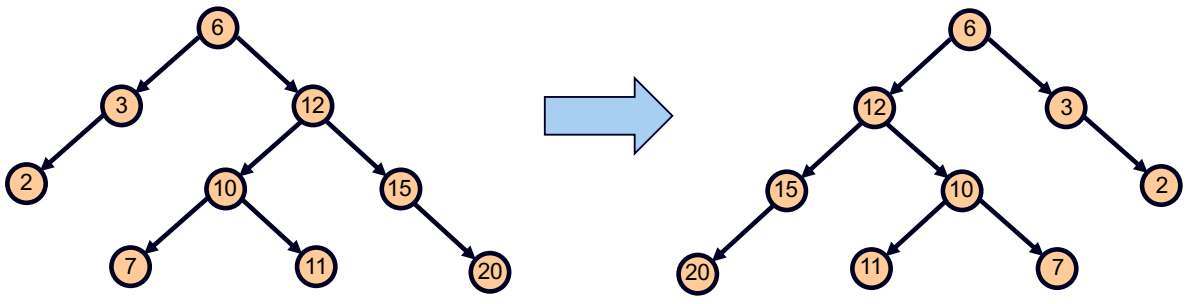
Exercício 1

- Somar valores pares de uma árvore de números inteiros

```
int CArvoreBinaria::somaPares(void){ // método public
   return somaPares(raiz); // devolve o resultado do método auxiliar
}
//---------------------------------------------------------------------
int CArvoreBinaria::somaPares(CNoArvoreBinaria *pArvore){ //método private
   int soma = 0; // acumulador da soma
   if (pArvore != nullptr) { // caso geral do algoritmo recursivo
      if (pArvore->dados % 2 == 0)       // se o valor no nó for par
         soma+=pArvore->dados;           // acumula o valor na soma
      soma += somaPares(pArvore->esq) +  // soma valores pares à esquerda
              somaPares(pArvore->dir);   //   e os valores pares à direita
   }
   return soma;   // devolve o resultado da soma
}
```


Exercício 2

- Transformar uma árvore binária de pesquisa na sua simétrica



Exercício 2

- Transformar a árvore binária na sua simétrica

```
template <class T>          // método public
void CArvoreBinaria<T>::tornaSimetrica() {
   tornaSimetrica(raiz);
}
//-------------------------------------------------------------
template <class T>          // método private (recursivo)
void CArvoreBinaria<T>::tornaSimetrica(CNoArvoreBinaria<T> *pArv) {
   // caso elementar do algoritmo recursivo: árvore/subárvore vazia
   if(pArv == nullptr) return;
   // troca o valor dos ponteiros das subárvores esquerda e direita do nó
   CNoArvoreBinaria<T> *aux;
   aux = pArv->esq;
   pArv->esq = pArv->dir;
   pArv->dir = aux;
   // torna simétricas as duas subárvores do nó *pArv
   tornaSimetrica(pArv->esq);
   tornaSimetrica(pArv->dir);
}
```